May 1979 Volume 7 Number 5

74

Popular Computing

The world's only magazine devoted to the art of computing.

	Decimal and/or alphabetic machines, intended for Business Data Processing.								Binary machines, intended for scientific and/or engineering computing.				
	1401	1410	1440	1620	7070	7074	7080	305	7090	7094	7040	7044	7030
LISP compiler													
PL/I compiler													
TYDAC interpreter													
SOAP assembler													
JOVIAL compiler													
IPL-V compiler													
SIMSCRIPT compiler													
ALGOL compiler													
COBOL compiler													
Fortran compiler													
Symbolic assembler													
Utility routines													

This entire issue is devoted to some recent history; namely, the introduction of the third generation of computers 15 years ago, and some of the effects of that significant event.

It is not likely that any one person could have first-hand knowledge of everything covered in this article. We will enjoy hearing of additions, corrections, and modifications.

Within the article, credit is given only to Fred Brooks. Due allowance should be made for the contributions of G. A. Blaauw and Gene Amdahl.

--Fred Gruenberger

Publisher: Audrey Gruenberger
Editor: Fred Gruenberger
Associate Editors: David Babcock
Irwin Greenwald
Patrick Hall
Contributing Editors: Richard Andree
William C. McGee
Thomas R. Parkin
Edward Ryan
Art Director: John G. Scott
Business Manager: Ben Moore

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 per year. For all other countries, add $3.50 per year. Back issues $2.50 each. Copyright 1979 by POPULAR COMPUTING.

@ 2023 This work is licensed under CC BY-NC-SA 4.0

THE BIG BYTE

The layout on the cover, which shall be referred to as Figure Z, shows the situation facing IBM in 1962. On the left is a partial list of the different computers that IBM was then making. The list is dichotomized into the two distinct breeds of machines that the industry had at that time; namely, the fast binary machines for scientific and engineering work, and the slow decimal (and/or alphabetic) machines for commercial data processing. The 305 defies categorization.

These machines were not completely different. The 1401, 1410, and 1440, for example, were logically the same, but differed in their frills and extras. The 1440 was faster than the 1401 and had more operation codes, but was otherwise the same logical device. That meant that within the 1400-series there was upward compatibility; that is, any program that would run on a 1401 would run unchanged on a 1410 or 1440. The reverse situation (that is, downward compatibility) was not true.

Similarly, the 7090, 7040, 7044, and 7094 were much the same machine. The 7030 ("STRETCH") was unique--one of IBM's little disasters.

Across the top of the layout of Figure Z is a selection of what is called support software; that is, software that was to be provided to the user "free" by IBM. To the left of the heavy line there is the software that was supplied; to the right of that line is a partial list of support software that the customers would like to have had.

Some of this software is relatively inexpensive to produce (an assembler, for example) and it is to the vendor's own advantage to produce it, since the vendor's own people must be able to use the machine and live with it, and they are no better at programming in absolute octal than you are. Moreover, in producing other software, they need close control of all of the machine's functions (including the pathological ones) and hence prefer (or at least did in 1962) to work in assembly language.

COBOL compilers, on the other hand, are very expensive to produce (particularly so in 1962). It was then only two years since the edict (by the U.S. Department of Defense) had been issued that demanded a COBOL compiler for any machine that was to be sold to the DOD.

"Utility routines" is a catch-all title that covers a multitude of programs necessary to market and maintain a line of machines. If the machine is binary, for example, it will require input and output routines to translate decimal (Hollerith) code into binary going in to the machine and binary to decimal going out. It would be grossly inefficient to have the users produce such routines. They might, to be sure, produce them better than the vendor, but it is to everyone's advantage to have such routines standardized so that other programs can be shared. Along the same lines, utility routines include such things as multiple precision subroutines, floating point subroutines, and elementary function subroutines, such as square root and trigonometric functions.

The chief utility routine is the diagnostic program. The customers are going to want to have their machines serviced, both for routine maintenance and in case of malfunctions. For this purpose, IBM operates an army of servicemen (yes, in 1962 they were all men) known as Customer Engineers--or CE's. Ideally, these men would be graduate electronics engineers with years of experience. In practice, they were usually high school graduates, paid very little, and the one you got was just recently hired. To help these people, IBM produced a program for each machine, called a diagnostic, that would put the machine through its paces and pinpoint, as far as possible, any trouble that was revealed.

For example, you want your machine to divide properly (this is true even if you have no program that uses division). The diagnostic program would perform, say, 10000 divisions on predetermined numbers and sum the quotients and check that sum against a predetermined value. If the two numbers agree, the program proceeds to its next test. If the two numbers disagree, then there is an indication of trouble in the divide circuitry, and the diagnostic should print an error message (assuming that the output circuitry of the machine was working) indicating which circuit card should be replaced. If every test proceeded without trouble, the program would print

MACHINE WORKS PERFECTLY,

at which point the CE, if well trained, would fold up his tool bag and head for the office, disregarding the fact that the machine might not divide with <u>your</u> data (or that smoke was coming out of the machine).

The flaw in this whole theory was that CE's quickly learned to tune their machines so that they would certainly pass the diagnostic, even if they would then do little else. Nevertheless, the concept of the diagnostic program makes it feasible to market machines by the thousands, using relatively low grade fixer-uppers. Each diagnostic program, which was quite long to begin with, got longer every week, as the reports trickled back to the factory of machine malfunctions that it would not catch.

All of this software, which appears as "free" to the person who rents the machine, costs money, and lots of it. A Fortran compiler cost at least $50,000 in 1962, and a COBOL compiler much more. But the initial cost of producing, debugging, testing, and documenting (for maintenance purposes) a piece of support software is only the smallest part of the expense. There must be reference manuals, for example, which must be kept up to date, and they are far from cheap. Most important, and most costly, is the maintenance of the programs. When the users report any sort of bug, such as...

(a) It says here in the manual that I can do thus and so, and here is a printout showing how I tried it and it didn't work...

(b) Here are two actions that each work fine except when I use them together...

(c) It says in the manual never to do this, but I forgot and did it anyway, and it seems to work fine...what goes on here?

(d) The logarithm subroutine gives excellent results except for the following peculiarity:

log .12345677 = -.9084850897

log .12345678 = +.9084850545

log .12345679 = -.9084850193

...then the maintenance team must check out the complaint (is the reported trouble true only in his machine or his compiler, or is it in ours, too?), fix it if valid, arrange to check out the fix in the next release of the compiler, and see to it that the pertinent reference in the manual is corrected. They should also arrange to check out the complaint against the other compilers. A user may have found a glitch in his Fortran (Release 23) for his 1440. It would be wise to look into Release 24 (just about to be shipped) and into the current Fortrans for all the other machines. It is not unlikely that the logarithm calculation is being repeated from compiler to compiler. It is easy to see how one tiny bug can tie up a lot of expensive people for a long time before it is finally conquered.

Writing support software, documenting it, maintaining it, distributing it--all this cost money. For the purposes of this discussion, let's say that each box in Figure Z cost IBM $100,000 per year. There are already entirely too many boxes in the matrix, but the situation was bound to get worse.

First, there was steady pressure from the customers to add to the number of free programs. To the right of the heavy vertical line are listed some of the compilers, assemblers, and interpreters that various organized groups of customers had requested, and the list is incomplete. The requests always show up in the form "If you would whip out a JOVIAL compiler for machine X, then we would order a trainload of X's." Such deals are tempting, but they have to be resisted. If a JOVIAL compiler appears for machine X, what can you say to the users (or potential users) of machine Y? The little boxes in matrix Z cannot be added by ones; they come in whole rows or columns.

But there was also pressure to increase the matrix in the other direction, and this pressure came from IBM's own people. It would appear like this: "If we took the 7070 and removed the DIVIDE op-code (who needs division?) and cut the core size and the cycle time, we could call the new machine the 7079 and we could sell hundreds of them to the wholesale vegetable industry." (The "we" in that sentence refers to some salesman, who would stand to make a huge sale.) But again, increasing the size of the matrix by adding a machine adds at least four of those little boxes, and that adds $400,000 per year in costs that must be recovered. The people who make such decisions thought twice at increasing that matrix in any direction.

At this point, along came Fred Brooks with an idea. Why not take the concept embodied in the 7040-7044-7090-7094 series of machines and extend it acreoss an entire new line of equipment? Why not make just one machine, essentially, in a wide range of sizes, to cover the complete spectrum of users' needs? If the whole line of equipment was logically the same (that is, had the same architecture), then there would be big savings in manufacturing, marketing, servicing, and all that support software (i.e., one Fortran compiler instead of 13 of them); the machines would all be naturally upward compatible.

Right there he had their complete attention. "Upward compatible" is a time-honored good thing in the IBM world, dating back 50 years or more to the days when new customers were lured into the fold with $10-per-month keypunches and a friendly service bureau that would process the cards that were punched. Each new customer, after a few months, would then be gently introduced to a sorter, and a reproducer, and a small tabulator, and in short order find himself looking at an acre of punched card machines, together with a monthly rental bill of $5000.

Consider the 1962 customer with a 1440, whose workload is heading rapidly toward saturation--this is, 168 hours of work for the machine each week--what are his choices? Short of improving his efficiency and squeezing more work out of his 1440 (this option seldom occurs to anyone), he can go one of two ways:

(a) Get a second 1440. This will double his machine rent and his floor space and his power and air conditioning, and significantly increase his payroll. It may actually increase his troubles. As a "solution" to his problems, it will certainly guarantee future more serious troubles.

(b) Switch to a larger and faster machine (say, the 7080). This will require a major conversion of every program; in ordinary English, that means rewriting every program in a new language. This is not only expensive, but terrifying, since there are a fair number of programs, vital to the running of the installation, that are completely undocumented and the customer doesn't even know <u>how</u> to convert them. The manager of the installation can't admit that, of course, so he finds endless reasons (all excellent) why a major conversion is unacceptable.

So now it was suggested that the entire IBM line be made upward compatible. This would mean that the customer facing saturation could be gently moved to the next larger machine, and the only change he would notice would be a small increase in his monthly machine bill. The new concept was well received and was given a code name, NPL, for "New Product Line." The NPL line of machines would be made in various sizes with model numbers

30 40 50 60 70 80

The model 30 would be for the kiddies: small, slow, with a short instruction repertoire, renting for around $5000 per month. The model 80, at the other extreme, would be blinding fast, would have lots of op-codes, and would come only in the large, economy size of storage, renting for, say, $150,000 per month--something for those serious about computing. All models would use the same logic, so that any program that would run on any model would run on any model with a higher model number.

But Mr. Brooks further suggested that the new product line be made also <u>downward</u> compatible. This was a neat trick.

Up to this time, a customer of IBM could obtain the complete logical design of any machine, right down to the most obscure details. As long as one restricted himself to machine language, it was possible to know and understand the complete operation of the machine.

Consider, now, the compiling and running of a Fortran program. The object code in in machine language, to be sure, but the programmer's thinking is at the Fortran level. In a very real sense, he is not dealing with a 7090, say, but with a different machine called Fortran. Between the programmer and the machine there is another program of some 50,000 instructions that he didn't write. It contains several thousand decisions, none of which are known to the user, each of which can affect his work.

The suggestion now was to apply that same concept to the new machine line, so that the user would be in this position:

$$\begin{tikzcd}
\boxed{\text{User}} \arrow[r] & \boxed{\text{Intermediate program}} \arrow[r] & \boxed{\text{NPL machine}}
\end{tikzcd}$$

That is, there would be a little program (somewhat akin to that Fortran compiler) interjected between the users and the new equipment. The new program would incorporate all the features then known under the term "monitor" plus lots of new control features, all adding up to what might be called an "operating system." It was conjectured that this intervening program might run to between 50,000 and 150,000 instructions and be cranked out in six months or so by a few good men.

{As it turned out, the program eventually ran to some 2,000,000 instructions, took six years to become fully operational, and kept some 250 experts busy. This part of the master plan was something of a disaster.}

With that intermediate program interposed between the users and the machines, it was no longer necessary or expedient to reveal all the logical details of the new equipment; in fact, quite the contrary. Suppose that a program is written using the XYZ op-code, and the program heads for execution on a model 80. The model 80 has the XYZ op-code, of course, and the instruction will be assembled and executed in the normal way. But now suppose that that program heads for a model 30, which does not include the XYZ op-code in its repertoire. The intervening program will then simply insert a subroutine into the program that performs the same function as the XYZ op-code. It may be that the user will not even be aware that this has been done; all that he will observe is that the XYZ operation functions properly on any machine. Thus, it can be said that the entire line will execute the same large set of codes, and the machines are magically downward compatible, or so it appears to the users. Since the new machines will operate significantly faster than the old ones, the observable operating speed will be greater even when the instructions involve subroutines.

The master plan had other novel features. The dichotomy among types of machines shown in Figure Z had to be cared for. The scientific/engineering users demand high speed of computation, with no great need for input/output sophistication. For that group, the new machines must be binary, since those customers were well trained in binary and were aware that the computational speed that they required was obtainable only that way.

On the other hand, the business data processing people will want fast and reliable input/output gear, and much prefer internal alphabetic and decimal capability. They were trained this way (by IBM); deep down, they don't really believe that arithmetic done in binary comes out quite right.

To satisfy these opposing viewpoints, it was suggested that the new line of machines be made word-oriented, with a 32-bit word.

At this point, we have to backtrack and talk about ancient architectures like character-addressable (as in the 1620, 1400-series, and the 7080) and variable-word-length. These were machine mutations which, among other things, permitted decimal arithmetic without the need for (and the nuisance of) multiple precision. On such machines, data words could be any length, from two decimal digits up to the storage capacity of the machine, with no wasted space in storage, and with no alteration in the instructions when the data changed.

The point is, the business data processing people, who were beginning to show signs of becoming the dominant group of computer users, had been carefully conditioned to such things, and had come to believe that they were natural. At one time, nearly all the business computing in the world was being done on IBM 1400-series machines (not to mention a small share that was done on IBM 1620's, which were supposed to be "scientific" machines).

So here we meet the byte. The 32-bit binary word would be divided into four 8-bit units, individually addressable, called bytes. In 8 bits there can be 256 bit-combinations, and these combinations would be the character set of the new machines.

To this day, there are large computers for which the internal character set is dictated by the IBM 026 keypunch (of 1948); the user must operate in a world limited to:

26 English capital letters;

10 decimal digits; and

12 special characters, such as: period, comma, plus, minus, virgule, asterisk, left and right parenthesis, dollar sign, quote, equals, and blank.

An internal character set of 256 characters could include lower case letters, Greek letters, all sorts of new special characters (such as arrows pointing in all four directions), with room left over. But in particular, there could be the decimal digits (stored essentially in binary-coded-decimal notation), for which op-codes could be furnished to perform decimal arithmetic. Words could be chained, using byte addressing, so that the net effect would be the same as that achieved by the old variable word length machines.

One more thing. The logical design that von Neumann proposed in 1946 had as one of its principle tenets the idea that data words and instruction words would have the same description and be stored in the same medium. The only distinction between the two would lie in their instantaneous function; if the word headed for the accumulator, it was then a data word; if it headed for the control circuits, it was then an instruction word. Indeed, the nub of what constitutes a computer, to distinguish it from other calculating devices, is precisely the ability of the same word to be both, at different times during the execution of a program.

It had already been demonstrated, particularly with the 1400-series machines, that instruction words and data words need not be the same length. The ill-fated OMIBAC of General Electric had (unsuccessfully, to be sure) demonstrated that they need not be stored in the same medium.

But it was fundamental to all machines that instructions could and should modify other instructions. The invention of index registers had shown that direct address modification was not essential. The new machines of IBM introduced another novel idea: that of base registers, which fostered address modification essentially through second-level addressing. The concept had first shown up on the CDC 160, but here it was being exploited much more extensively.

Where are we? We have a line of machines with identical logic, upward and downward compatible, designed to suit the entire spectrum of users. Once committed to this new path, there remained such tasks as putting it all into production and selling it to the customers. The gag line at the time was "IBM is playing a game called You Bet Your Company, and with very small chips" (an oblique reference to the most visible difference in the new hardware; namely, integrated circuitry, fabricated on what seemed then like tiny circuit boards).

The new equipment was announced in April, 1964. There was a pre-announcement announcement of such a major event, so that a major portion of IBM's customers assembled one day that April to get the word.

They all got it simultaneously. The announcement was prepared as a color 16 mm movie, and every IBM office was furnished with two prints. The films were started at 9:00 A.M. in Los Angeles, 10:00 in Denver, 11:00 in Chicago, and noon in Washington; no one would have a time advantage in ordering the new equipment. The message being communicated was this: "You know all that marvelous equipment we sold you?--it's now junk. We'll show you the new wave, and you ought to join it. The boys in the back will take your orders. First come, first delivered; stock up before the hoarders get busy."

The customers weren't really shown. The film purported to show an operating installation of the new equipment (now yclept System/360), but the gadgets in the film were dummy boxes, cleverly photographed to look like functioning 360's. At the time the film was made, there were no functioning 360's, at least not fit to be seen. Nevertheless, the film made a gorgeous impression. It said very little about the new gear, but was emphatic that it was good. It is said that the Los Angeles IBM office alone had orders for 1000 machines before the first week was over.

That was April, 1964. First deliveries of the 360's would be at the end of 1965. Several interesting things happened in those 18 months.

The 1130 appeared--a small machine, whose logic was pure binary--definitely not part of the 360 line--forced out by strong competition.

Then the 360/44 was announced, in response to demands for a "scientific" machine. The 44 was very 360-ish, but not fully compatible with all the rest of the line.

The big event was the announcement by two old and influential customers--Bell Telephone Laboratories and M.I.T.--that the new line didn't fit their needs and they were ordering competitive equipment.

One has to look at this situation from IBM's viewpoint, which is that the computing business is 100% theirs by divine right, less that portion that they graciously share with others. They can tolerate a certain amount of defection to the enemy, but the action of two prestigious customers in rejecting the new offering hurt, especially since those two customers chose to go to General Electric for their machines. A short time later, the 360/67 popped out, which turned out to be exactly the machine that Bell Labs and M.I.T. had wanted. The 67 was the "timesharing" version of the 360, and it was also not completely compatible with the rest of the line.

A Model 90 was announced; this was IBM's third unsuccessful attempt to produce a super computer (the others were NORC and STRETCH). The 90 was pulled back quickly, before anyone could place a firm order for one, and the Model 91 soon appeared. The 91, too, was to be withdrawn, but not before several customers had placed firm orders, and IBM was forced to build a few of them. (It was less than a decade since IBM had entered into a consent decree that specified, among other things, that they could not announce a machine that they had not already built, and that they must deliver any machine that was announced, for which someone placed a firm order.) Finally, the top of the line became the Model 195, which was moderately successful.

In November of 1964, RCA announced their new line, called Spectra 70, to be made in models numbered

35   45   55   65   75

RCA arranged to notify all their customers simultaneously, just like IBM, but through closed circuit television instead of 16 mm movies. The effect wasn't quite the same; film can be edited and polished and re-shot. Television can have people's faces all purple, and does, especially at the most critical time, like when RCA's president was describing their new equipment.

RCA copied something else, too; namely, the complete logic of the 360. They made no bones about it: "We like IBM's new design, and we're going to make the same machines." It was implied by their numbering system that RCA would furnish about the equivalent of a 360/40 at the price of a 360/30, and deliver the machines sooner than IBM.

Notice that this was the first good indication that IBM had that their new line might be successful. There had been lots of orders, but no deliveries as yet; in fact, there were no production models finished at this time. Suppose the customers didn't like the new machines? There are few parallels in industrial history to such a giant gamble. But here was RCA not only endorsing the plan, but latching on to it. (RCA did deliver Spectra 70's; they were fine machines, and some of them are still in operation, although RCA has long since left the computer business.)

The 360's constituted the third generation of computer equipment. Nearly every other manufactuer of large mainframes adopted the logic of the 360's, albeit not as blatently as had RCA. The only big holdout to the scheme of architecture was Control Data, who decided to go with pure binary for one more round at least. Thousands of 360's were sold and eventually dominated the large computer market, both in the U.S. and abroad.

There was another interesting aspect to the introduction of the 360's: the mass conversion that it triggered. Almost all the business data processing was being done on the 1400's, as we have mentioned, and the 360 announcement meant that the 1400's would now be phased out. IBM has always had the policy that a customer could keep any machine as long as he pleased--provided that he paid the rent--and IBM would maintain it. Thus, the 10,000 or so users of 1400-series machines did not have to panic; they could keep their machines indefinitely if they wished. Eventually, of course, spare parts would become scarce, as would top-notch service.

These users were pulled two ways. The argument in favor of switching machines was that the replacement (the 360/30) cost no more in rental, used the same amount of floor space and power, and ran about 5 times as fast. If the 1400 user was running around the clock (which is four shifts of people), he could then, in theory, get his work done in less than one shift, and thus save considerable money and time.

The argument the other way (to stay with the 1400's as long as possible) was that major conversion. It is difficult to imagine two more different machines than the 1401 and the 360/30. Some day it might be possible to convert from one machine language to another by means of a computer program, but at that time the only possibility was to reprogram everything. Of course, the conversion couldn't be put off forever, in any event.

At best, the distance in time between ordering a 360 and having it delivered is about 18 months. This time delay allows IBM to predict their production schedules with precision. Even better, it gives their customers time to learn about the new machines and get ready for them. IBM has learned that if they give someone a new machine when he asks for it, there is a distinct possibility that he may be unprepared, but use it anyway. Then if the scheme doesn't work out, he tends to blame IBM.

So each customer who made the switch had 18 months in which to re-do every program in his library, and to debug and test those programs. For this purpose, he needs access to a 360/30. He could have that, furnished free by IBM, at their service centers. The center might not be too handy, and the free time might be from 2 to 6 A.M., but it was one way to go.

Another way was to utilize a program that IBM furnished, which would run on a 1401 and simulate the actions of a 360/30 in every detail. This would be fairly efficient; it is easy to simulate a fixed-word-length binary machine on a 1401.

When the delivery day arrives, the machines are swapped on a weekend, and the IBM people spend that weekend checking out the new machine. On Monday morning, there is the new equipment, fully operational, and over there are the filing cabinets full of checked out card decks. There has been no doubled rental.

...All except the dozen or so programs that have not been converted. In some cases this came about simply due to a shortage of time. In many cases, however, it came about for the reason stated earlier; namely, that no one knew how to rewrite the program, because no one knew its logic or anything about it, except that it worked.

To help this situation, IBM now had another simulation program, this one to simulate a 1401 on the 360. This program is much more difficult to write than the other one, and it runs much slower. In fact, for 1401 programs using multiplication and division on long data words, the simulation would be really slow, to the point where the customer's work would suffer.

A new problem...and another brilliant solution at hand. For a small sum, like $40 a month, a small sealed box can be rented and attached to the 360 to speed up the simulations; it is called an emulator. It speeds up multiplications and divisions electronically, and one might suspect that it is made of old 1401 parts, of which there must be plenty. In any event, it allows the user to buy his way out of his troubles. After a few months go by, and he reprograms those last few jobs, the gadget can be disconnected and returned to IBM. The following is conjecture: perhaps half of the 10,000 installations who made the move from the 1401 to the model 30 took the emulator package. The 360/30's ran from 1965 to 1975 overall; perhaps the average life was around 5 years. The bulk of those 5000 emulators were rented for most of the 5 years.

After the first few months, most of the installations probably no longer knew that they were renting the little box; a $40 item hardly stands out on a lengthy bill that totals $5000. But even if the installation manager was aware of the item, what were his choices? Each month he can decide to face facts, find out what those programs do, and assign someone to rewrite them. Or, for $40 he can put the whole problem off for another month.

Shortly after the 360 was announced, a group in the SHARE organization came up with an idea to do for software what the 360 would do for hardware; namely, eliminate that dichotomy of use. The scientific and business users of computers not only have a different view of machines, but also of how to program them. They speak in different tongues--namely, Fortran and COBOL.

SHARE is the original users group, started at the time the 704 was about to arrive. The idea they came up with was to produce a new language that would combine the best features of both Fortran and COBOL while eliminating the faults of each of the old languages. They suggested a joint project between SHARE members and IBM to produce this language for the new machines (actually, since compilers are supposed to be machine-independent, the language would only be specified; compilers could then be written for any machine).

IBM liked this idea, and offered to put up $150,000,000 in support. They gave it a code name: NPL, for "New Programming Language." They're not too imaginative about things like that at IBM.

When the project was over, several years later, the language was formally named PL/I and the total cost had risen to something like $180,000,000. It was presented to the industry with great fanfare and it was expected to generate the same interest and enthusiasm as had the 360 announcement. It was surely clear to IBM that their experience with Fortran a decade earlier was about to be repeated. There were strong hints that support for Fortran and COBOL was to be phased out, and rather rapidly. As with the 360, the message was "Here is the new wave--get with it."

The response from the industry was less than enthusiastic. A few installations converted all their operations to PL/I; books on it were duly produced; the usual fanatics proclaimed it as the last word in magic languages--the millennium was here. On the whole, the reaction was deadly.

In many installations, a few programmers took on the task of trying to convince their management that PL/I was the way to go. In all fairness, their arguments were sound and logical. The language _is_ an improvement over earlier languages; one language in the shop is much better than two; programmers can talk to each other or even act interchangeably; and so on. Young people can never understand why the older generation doesn't rush to embrace each new notion.

The installation managers resisted the PL/I juggernaut nobly. Their stand was one or more of the following arguments:

(1) Who needs it? Fortran rushed in to fill a vacuum; there was nothing around in 1956 but absolute binary, or assembly language (Speedcode, or perhaps SAP). The appearance of the first real compiler was a sensation. The early Fortrans were inefficient and full of bugs, but they were novel, easy to learn and use, and they offered the first break from a rigid tyranny. Programmers couldn't wait to show off proficiency in the new art. "Look, Charlie, all the red tape parts of a loop can be specified in one simple canonical statement"...it was intoxicating and contagious. This was the phenomenon that IBM fully expected to repeat.

But there was now no vacuum; Fortran and COBOL were both here and working fine. There was little excitement offered in learning PL/I; in fact, since the designers had produced an exceptionally rich and elaborate language, it looked like it might be hard work to learn it fluently. In short, there was not only no real pressure to change, but some pressure to avoid changing. Vague future benefits (few of which could be guaranteed) were not enough to surmount this argument.

(2) It's IBM. There were two groups who resisted PL/I because of its IBM origins: IBM users and non-IBM users. People get a psychological "set" about vendors; you might consider your own attitude toward the various auto manufacturers, or camera makers, or the TV networks, or whatever. However it came about, one argument advanced against adoption of PL/I was its parentage.

(3) We'll wait for the bandwagon. This is the argument that points out that the cost of pioneering is usually high; that pioneers have been known to become martyrs; that we'll let the other guy try it first and see how he fares. Meanwhile, we'll just muddle along the old tried and true way.

(4) We must protect our investment in operating programs. Our library of such programs represents X millions of dollars of programming effort, and we have to keep those programs current and functioning. That is, we must employ expert Fortran and COBOL programmers anyway, so we might as well let them write new programs, too. If we go to PL/I, we might just wind up with three kinds of expensive people rather than the one group being touted.

(5) We have to think of the labor pool. If one of our Fortran programmers leaves us for any reason, we can obtain a replacement within a day; the town is full of Fortran veterans and we can have any number of them simply by offering a high enough salary. The same is true of COBOL experts, many of whom are also familiar with our particular type of business. The labor pool is active, large, and viable, and its presence protects us. There is no fluent labor pool in PL/I; in fact, PL/I experts seem to be quite scarce (see item (3) above) and unduly expensive. If we convert our entire operation to PL/I, we will be the labor pool and this is not a good position to hold.

Note: Each of the above reasons is almost totally independent of PL/I; you could follow the arguments without knowing anything at all about the language (a state of affairs that was true for most of the managers who turned it down). Some of the arguments are even slightly irrational and emotional. But humans make important decisions on just such grounds.

The last reason is not so imaginary, and is much harder to refute:

(6) PL/I must be inherently inefficient. If a language contains the capability of both Fortran and COBOL, and I want to use it for a pure Fortran-type problem, then all the COBOL-type elements in the language are going to cost me.

Any general-purpose tool is inefficient compared to all the special-purpose tools it replaces. You can turn a hex nut with either of two tools: an open-ended hex wrench that exactly fits the nut (and you'll need a set of such wrenches), or an adjustable crescent wrench that can be made to fit any nut, but never fits any of them as well as the fixed head wrench. You swap efficiency for convenience. Seldom can you have both at once.

Indeed, the first thing a manager notices about an intended conversion to PL/I is that his machine needs more core storage, since the PL/I compilers demand it. Is the cost of that extra core a hardware or software cost?

This situation was not helped by the fact that the early PL/I compilers were not well written and were inherently inefficient. Later versions were much better, but the problem still remains. From time to time, someone makes measured comparisons as an attempt to refute the "inefficiency" argument, but almost always they compare the wrong things. For example, they compare one PL/I compiler to a previous one, to show the tremendous improvement in speed (of compilation, or execution, or whatever). The comparison that sceptical managers make is the one between a PL/I compiler and the corresponding Fortran and COBOL compilers. To make this comparison properly, one would have to have the same task (which should be a non-trivial task) coded at least twice, with whatever it is that is valued carefully measured; this might be programmer time, or compilation time, or number of needed debug shots, or execution time. For a valid A-B test, the task would have to be coded and measured at least four times. Few people have the time or ambition to conduct such tests properly, so the basic problem hangs there untouched.

The reasons given above may not be wholly rational or logical, but they are certainly real. If IBM had simply named the language Fortran VI, we would probably all be using it now. PL/I was announced in 1965 and it has slowly withered away ever since. Today, no one seems to be advocating it, and it has joined the graveyard inhabited by ALGOL, JOVIAL, COMTRAN, and SIMULA. Requiescat in Pace.

Although PL/I was not successful, it did have a place in the area of computer language development. Other aspects of the 360 epic were also less than successful.

For example, it was thought that the number of Fortran compilers could be cut down when there were less machine types, but reality went the other way. Each of the 360 models was available in a wide range of sizes, and the customer who paid for a large one (say, with three times as much core storage as the minimum configuration) wanted a Fortran that capitalized on that much storage. Thus, Figure Z did not reduce to one cell per compiler as had been anticipated; in fact, there were probably more Fortran compilers for the 360's than there had been for the earlier machines.

Thousands of 360's were installed and they became the standard of the field. The shift to System/370, which began in 1971, was a very quiet affair by comparison, although it may eventually mark the introduction of the 4th generation of computers (we usually note the demarcation between computer generations only in retrospect).

● ●

From time to time, Professor Richard Andree (University of Oklahoma) unleashes a small battery of inconsequential problems on the world (the last batch we had appeared in issue number 16). This batch, except for the last problem, is intended for the first few weeks of a computing course for beginners "who think they know how to compute either because they took a one-hour course in Fortran or Algol or because they learned BASIC in high school."

CHALLENGE problem for week (1)

Which is larger, π^e or e^π? Justify your answer.

CHALLENGE problem for week (2)

Determine the smallest value of N such that N! contains

a) All nine of the non-zero decimal digits
b) All ten decimal digits.

Justify your answer. .

CHALLENGE problem for week 3

Step 1 Let N_o be a positive integer.

Step 2 Let S_k = the sum of the squares of the digits of N_k.

Step 3 Let $N_{k+1} = S_k$

Step 4 Increase k by 1.

Step 5 Go to Step 2.

For example, if $N_o = 86$, $S_o = 64 + 36 = 100 = N_1$;
$S_1 = 1 + 0 + 0 = 1 = N_2 = N_3 \ldots$

What happens to the sequence of N's for various possible starting values N_o? Make a conjecture and prove or disprove that conjecture.

CHALLENGE problem for week 4

Find the smallest positive ($\epsilon > 0$) floating point number that can be added to 1 without producing $1 + \epsilon = 1$ in the computer and language you know best.

CHALLENGE problem for week 5

If $N! = 1 \cdot 2 \cdot 3 \cdots N$

and $S(K) = \sum_{N=1}^{K} (N!) = 1! + 2! + 3! + \ldots + K!$

then for what positive integers K is S(K) a perfect square? Justify your statement.

For example:

If K = 3, S(K) = 1! + 2! + 3! = 9, is a perfect square.

If K = 4, S(K) = 1! + 2! + 3! + 4! = 33, <u>not</u> a perfect square.

...and a more difficult CHALLENGE problem:

Find square matrices B such that

$$B^2 = \begin{bmatrix} -91 & 28 & 9 \\ 47 & -14 & -4 \\ -1113 & 341 & 108 \end{bmatrix} = A$$

How many such matrices exist?
We wish <u>exact</u> square roots, not ones such that B^2 is close to A, since the latter may be <u>very very</u> far from the true B's.